



PROJECT REPORT

On

NETWORK INTRUSION DETECTION USING NOVEL DSSTE

Submitted in partial fulfilment for the award of degree

Of

Bachelor of Technology

In

Computer Science & Engineering

By

ASIF MAJEED (MLM20CS045)

Under the Guidance of

Mr. JAYAKRISHNAN B

(Assistant Professor, Dept. of Computer Science and Engineering)



**DEPARTMENT OF COMPUTER SCIENCE&ENGINEERING
MANGALAM COLLEGE OF ENGINEERING, ETTUMANOOR**
(Affiliated to APJ Abdul Kalam Technological University)

MAY 2024



MANGALAM COLLEGE OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Vision

To become centre of excellence in computing and research where future generations embrace technologies wholeheartedly and use its possibilities to make the world a better place to live.

Mission

Enlight the young talents to achieve academic excellence as well as professional competence by imparting state of the art knowledge in computing and to be admirable individuals with ethical values and appropriate skills.

Program Educational Objectives

PEO 1: Graduate will have strong foundation and profound knowledge in computing and allied engineering and be able to analyze the requirements of real world problems, design and develop innovative engineering solutions and maintain it effectively in the profession.

PEO 2: Graduate will adapt to technological advancements by engaging in higher studies, lifelong learning and research, there by contribute to computing profession.

PEO 3: Graduate will foster team spirit, leadership, communication, ethics and social values, which will lead to apply the knowledge of societal impacts of computing technologies.

Program Specific Outcomes

PSO 1: Apply the Principles of Computing in solving real world problems with sustainability.

PSO 2: Apply Futuristic technology in designing and developing hardware and software solutions.

MANGALAM COLLEGE OF ENGINEERING, ETTUMANOOR
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

MAY 2024



CERTIFICATE

*This is to certify that the Project titled “Network Intrusion Detection using novel DSSTE” is the Bonafide record of the work done by Asif Majeed (MLM20CS045) of B.Tech in Computer Science and Engineering towards the partial fulfilment of the requirement for the award of the **DEGREE OF BACHELOR OF TECHNOLOGY** by APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY, during the academic year 2023-24.*

Internal Examiner

External Examiner

Project Guide

Mr. Jayakrishnan B

Assistant Professor

Department of CSE

Head of the Department

Ms. Neema George

Professor

Department of CSE

ACKNOWLEDGEMENT

I am greatly indebted to the authorities of Mangalam College of Engineering for providing the necessary facilities to successfully complete my Project on the topic “**Network Intrusion Detection Using Novel DSSTE.**”

I express my sincere thanks to **Dr. Vinodh P Vijayan**, Principal, Mangalam College of Engineering for providing the facilities to complete my Project successfully.

I thank and express my solicit gratitude to **Ms. Neema George**, HOD, Department of Computer Science & Engineering, Mangalam College of Engineering, for her invaluable help and support which helped me a lot in successfully completing this Project work.

I express my gratitude to my Internal Guide, **Mr. Jayakrishnan B**, Assistant professor, Department of Computer Science for the suggestions and encouragement which helped in the successful completion of my Project.

Furthermore, I would like to acknowledge with much appreciation the crucial role of the faculties especially Project coordinator, **Mr. Jayakrishnan B**, CSE Department, Mangalam College of Engineering, who gave the permission to use all the required equipment and the necessary resources to complete the presentation & report.

Finally, I would like to express my heartfelt thanks to my parents who were very supportive both financially and mentally and for their encouragement to achieve my goal.

ASIF MAJEED (MLM20CS045)

ABSTRACT

In the era of rapidly evolving cyber threats, securing network infrastructures against intrusions has become paramount. This paper proposes a novel approach for network intrusion detection leveraging Dynamic Subsequence Space-Time Embedding (DSSTE). DSSTE integrates spatial and temporal information from network traffic data, offering a comprehensive perspective for anomaly detection. Unlike traditional methods that rely solely on static features, DSSTE captures the dynamic behaviour of network traffic, enhancing detection accuracy and adaptability to evolving attack strategies. Through experimental evaluation on benchmark datasets, our approach demonstrates superior performance in detecting various intrusion types while maintaining low false positive rates. The results highlight the efficacy of DSSTE in enhancing network security and its potential for deployment in real-world environments.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	List of Figures	i
	List of Tables	ii
	List of Abbreviations	iii
1	INTRODUCTION	1
	1.1 Background	1
	1.2 Introduction	2
	1.3 Problem Statement	2
	1.4 Motivation	3
	1.5 Scope	4
2	LITERATURE REVIEW	5
	2.1 Intrusion Detection of Imbalanced Network Traffic Based On Machine Learning and Deep Learning	5
	2.2 A Novel PCA-firefly based XGBoost classification mode l for intrusion detection in networks using novel GPU	6
	2.3 Siam-IDS Handling class imbalance problem in Intrusion Detection System Using Siamese Neural Network	6
	2.4 Increasing the performance of Machine Learning-Based on an imbalanced an Up-to-date dataset	7
	2.5 A network intrusion Detection system for Building automation and control system	8
3	PROPOSED SYSTEM	10
4	METHODOLOGY	12
	4.1 Anomaly Detection Model Projection	12
	4.2 Distance calculation	12
	4.3 Anomaly Thresholding	12
	4.4 Evaluation and validation	12
	4.5 Optimization and Finetuning	12
5	SYSTEM ARCHITECTURE	13
6	MODULES	14
7	DIAGRAMS	16
	7.1 Level 0 DFD	16

	7.2 Level 1 DFD	16
	7.3 Level 2 DFD	17
8	TESTING	18
	8.1 Unit Test	18
	8.2 Integration Test	19
	8.3 Performance Test	19
	8.4 Acceptance Test	20
9	ADVANTAGES & DISADVANTAGES	21
10	RESULTS	24
11	CONCLUSION & FUTURE SCOPE	26
	APPENDICES	28
	REFERENCES	44

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
Fig 5.1	System Architecture	13
Fig 7.1	Level 0 DFD	16
Fig 7.2	Level 1 DFD	16
Fig 7.3	Level 2 DFD	17
Fig 8.1	Unit Testing	18
Fig 8.2	Unit Test Output	19
Fig 10.1	Random Forest Feature Selection	24
Fig 10.2	XGBoost Selecting Best Feature	24
Fig 10.3	Bar Plot of Attack Types	25

LIST OF TABLES

TABLE NO.	TITLE	PAGE NO.
2.1	Literature Review	10

LIST OF ABBREVIATIONS

ABBREVIATION	FULL FORM
ML	Machine Learning
DSSTE	Dynamic Subsequence Space-Time Embedding
DL	Deep Learning
NIDS	Network Intrusion Detection Systems
R2L	Remote to Local
BACS	Building Automation and Control Structures
SMOTE	Synthetic Minority Oversampling Technique
IDS	Intrusion Detection Systems
PCA	Principal Component Analysis

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Deep learning (DL) and Machine learning (ML) have revolutionized cybersecurity by offering advanced capabilities in various domains. DL models can analyse network traffic, code, and user behaviour to detect intrusions, malware, vulnerabilities, phishing attempts, and fraudulent activities. They learn patterns and anomalies to provide accurate threat detection and enable proactive defence mechanisms. Machine learning has emerged as a powerful tool for addressing the challenges of intrusion detection in imbalanced traffic. Various techniques have been explored to mitigate the impact of imbalance and improve the accuracy of attack detection. With the rapid expansion of the internet and its integration into critical infrastructure, network security has become paramount.

Network intrusion detection systems (NIDS) serve as the frontline defence against malicious attacks, constantly monitoring network traffic and identifying potential threats. However, the ever-evolving landscape of cyber threats presents significant challenges to NIDS effectiveness. One such challenge arises from the inherent imbalance in network traffic, where normal traffic significantly outnumbers malicious attacks. This imbalance can significantly hinder the performance of traditional NIDS, often leading to high false alarm rates and inaccurate detection of actual attacks. Traditional machine learning algorithms tend to favour the majority class (normal traffic) during training, resulting in poor performance on the minority class (attacks). This leads to high false positives, where legitimate traffic is misclassified as attacks, and false negatives, where actual attacks go undetected.

The Intrusion detection in imbalanced network traffic poses a significant challenge to cybersecurity. However, recent advances in machine learning and deep learning offer promising solutions to address this issue. By leveraging advanced data augmentation techniques, cost-sensitive learning algorithms, and powerful deep learning models, researchers are developing more effective and robust NIDS capable of accurately detecting attacks even in the presence of significant imbalances.

As the field continues to evolve, we can expect even more sophisticated and intelligent intrusion detection systems that can effectively protect our critical infrastructure from ever-evolving cyber threats. This paper researches device mastering and deep studying for intrusion detection in imbalanced community site visitors. It proposes a novel DSSTE (algorithm to tackle the class imbalance hassle).

1.2 INTRODUCTION

In today's interconnected digital landscape, network intrusion poses a significant threat to the security and integrity of information systems. With the increasing sophistication of cyber-attacks, traditional intrusion detection systems (IDS) often struggle to keep pace with evolving threats. Static signature-based approaches are limited in their ability to detect novel intrusion patterns, while anomaly-based methods may suffer from high false positive rates. To address these challenges, novel approaches that leverage advanced techniques are imperative. In this study, we propose a pioneering method for network intrusion detection utilizing the Dynamic Subsequence Space-Time Embedding (DSSTE) algorithm.

The exponential growth of network traffic and the complexity of modern cyber threats necessitate innovative solutions for intrusion detection. Traditional methods often focus on static features, such as predefined signatures or simple statistical metrics, which may fail to capture the dynamic nature of network behaviour. The DSSTE algorithm offers a unique approach by integrating spatial and temporal information from network traffic data. By embedding subsequence of data into a dynamic feature space, DSSTE provides a holistic representation of network behaviour, enabling more effective detection of both known and novel intrusion patterns.

One of the key advantages of the proposed DSSTE-based approach is its ability to adapt to evolving attack strategies. Unlike traditional IDS that rely on static rules or signatures, DSSTE captures the dynamic behavior of network traffic, allowing for real-time analysis and detection of anomalous patterns. This adaptability is crucial in the face of constantly evolving cyber threats, where attackers continuously modify their tactics to evade detection. Additionally, DSSTE offers improved detection accuracy compared to traditional methods, thereby reducing the risk of false positives and ensuring prompt identification of potential intrusions.

The effectiveness of the DSSTE-based intrusion detection system is evaluated through comprehensive experimentation and validation. Benchmark datasets and real-world network traffic data are used to assess the system's performance in terms of detection accuracy, false positive rate, and computational efficiency. Furthermore, optimization techniques are applied to fine-tune various parameters of the system, enhancing its overall effectiveness and scalability. Through rigorous evaluation and optimization, we aim to demonstrate the efficacy of the proposed approach in enhancing network security and mitigating the risks associated with network intrusions.

1.3 PROBLEM STATEMENT

Intrusion detection structures (IDS) play a essential function in safeguarding community protection by way of identifying and preventing malicious activities. but conventional IDS techniques face

challenges when handling imbalanced network site visitors, in which normal traffic considerably outweighs attack visitors. This imbalance poses a extensive obstacle to accurate intrusion detection, as minority attack samples often get overshadowed by means of the overwhelming majority of everyday samples, leading to: Underfitting: machine gaining knowledge of models educated on imbalanced datasets tend to prioritize learning dominant styles in most of the people class (regular visitors). consequently, they underfit the minority class (attack site visitors), resulting in high false-terrible charges and ignored assaults. Biased selection limitations: The inherent bias toward the general public class in imbalanced datasets can lead to biased decision obstacles in the trained version. This bias translates to incorrectly classifying genuine attack times as ordinary visitors, compromising the general effectiveness of the IDS. Difficulty in learning complicated attack patterns: attack visitors frequently famous complex and nuanced styles which can be hard for traditional device gaining knowledge of fashions to seize, in particular whilst outnumbered with the aid of normal visitors. This difficulty hinders the capability of the IDS to discover novel and sophisticated assaults. Decreased average Accuracy and performance: The combined results of underfitting, biased choice boundaries, and issue in getting to know complicated attack patterns significantly impact the overall accuracy and overall performance of IDS in imbalanced community traffic situations. this may lead to improved protection dangers and vulnerabilities for networks beneath assault. This studies goals to research and tackle this assignment with the aid of exploring the ability of system studying and deep studying techniques to improve intrusion detection accuracy in imbalanced network visitor's eventualities.

1.4 MOTIVATION

Network intrusion detection structures (NIDS) are essential tools for shielding networks and structures from malicious attacks. conventional NIDS generally relies on signature-based detection, which identifies assaults through matching recognized patterns. But, this approach is confined in its capability to stumble on novel attacks and zero-day threats for which signatures have now not yet been evolved. Machine learning (ML) and deep learning (DL) provide promising options for intrusion detection. these strategies can robotically study complicated styles from network visitors' information, permitting the identity of unknown attacks. however, a prime undertaking in making use of ML and DL to NIDS is the inherent imbalance of community visitors' facts. Network traffic typically includes a considerable majority of normal visitors and a small minority of assault visitors. This imbalance poses a great assignment for ML and DL fashions, as they frequently tend to overfit the majority magnificence (regular site visitors) and fail to accurately detect the minority elegance (assault site visitors). *Network Intrusion Detection Using Novel DSSTE*. This paper addresses the essential difficulty of imbalanced network site visitors in the context of intrusion detection through the use of ML and DL. We

advocate a unique approach referred to as the Dynamic subsequence space time embedding (DSSTE) that makes a specialty of balancing the statistics by compressing most of the people's class and augmenting the minority elegance in hard samples. This technique ambitions to improve the accuracy of intrusion detection by offering the version with extra data approximately the minority magnificence and lowering the prejudice closer to most people magnificence. This research has the ability to noticeably enhance the effectiveness of intrusion detection and beautify the security of networks and structures. by using addressing the issue of imbalanced information, we will ensure that ML/DL fashions are able to correctly detecting both known and unknown assaults, contributing to a more secure and greater at ease virtual world.

1.5 SCOPE

The purpose of the new system is to investigates the effectiveness of DSSTE in improving the performance of NIDS in imbalanced network traffic scenarios. The key objectives are Evaluate the performance of DSSTE in terms of reducing data imbalance and improving the accuracy of intrusion detection. Compare the performance of DSSTE with existing techniques for imbalanced data learning. Analyze the impact of different hyperparameters on the performance of DSSTE. Investigate the effectiveness of DSSTE for different types of network attacks. By addressing the challenge of imbalanced data in network intrusion detection, this research contributes significantly to the advancement of cybersecurity and paves the way for more robust and effective NIDS solution.

CHAPTER 2

LITERATURE REVIEW

2.1 Intrusion Detection of Imbalanced Network Traffic Based on Machine Learning and Deep Learning

Key Aspects:

This paper the utilization of an ESP32-CAM board and a robot chassis in a surveillance robot system enables the creation of a mobile surveillance device. The ESP32-CAM board, known for its affordability, integrates a compact camera module and Wi-Fi connectivity. By incorporating a robot chassis, the device gains the ability to move and record video in various locations. The ESP32-CAM board hosts a web interface that allows users to control the robot's movement, access live video streams, and capture snapshots of the video feed. Moreover, the system can be programmed to detect motion using computer vision algorithms, such as object detection and tracking, and promptly notify the user. The surveillance robot utilizing ESP32-CAM exhibits promising applications in home security, remote location monitoring, and industrial surveillance. Its cost-effectiveness and user-friendly interface make it a convenient solution for individuals requiring remote surveillance capabilities.

The field of surveillance robotics has experienced significant growth in recent years, as it has found applications in various sectors such as security, industrial monitoring, and home automation. One notable project in this area is the development of a cost-effective and efficient surveillance robot using the ESP32-CAM. In today's modern era, surveillance robots have gained popularity due to their capability to remotely monitor and gather information from a distance. The integration of the ESP32 CAM module, which combines a camera module with Wi-Fi and Bluetooth connectivity, has enabled the creation of surveillance robots that can be controlled from afar.

The ESP32 CAM module is a compact camera module that can be seamlessly incorporated into a robot. It offers high-quality imaging capabilities and can be easily controlled using the ESP32 microcontroller. Moreover, the module's Wi-Fi and Bluetooth connectivity facilitate smooth communication with a remote-control station or a computer. By utilizing the ESP32 CAM module, it becomes feasible to develop a surveillance robot that can be remotely operated from a computer or a mobile device. This robot can be programmed to navigate a specific area and capture images or videos of its surroundings. The captured media can then be transmitted wirelessly to a remote-control station, where it can be viewed and analyzed in real-time. The integration of the ESP32 CAM module into

surveillance robots bring numerous advantages. These include the ability to capture high-quality images and videos, remote control functionality, and wireless data transmission.

2.2 A Novel PCA-firefly based XGBoost classification model for intrusion detection in networks using novel GPU

Key Aspects:

This paper presents a novel intrusion detection model for networks, utilizing a hybrid approach combining Principal Component Analysis (PCA), Firefly Algorithm, and XGBoost classification. The model leverages the computational power of GPUs for accelerated training. Through dimensionality reduction and hyperparameter optimization, the proposed model achieves superior accuracy, sensitivity, and specificity compared to existing methods. Additionally, the GPU implementation significantly reduces training time, making it suitable for real-time deployment. The paper highlights the advantages of the proposed model, including improved performance, reduced cost, and scalability. It also acknowledges challenges and outlines future research directions, paving the way for further advancements in network intrusion detection. Conventional community intrusion detection structures rely upon signature-based tactics, which can be confined to their ability to discover novel attacks. machine getting to know offers promising alternatives, however, excessive-dimensional datasets pose demanding situations. This paper addresses those challenges using offering a hybrid model combining 3 techniques: Principal component analysis (PCA): This dimensionality discount approach reduces the number of capabilities within the statistics, minimizing computational value and enhancing version performance. Firefly set of rules: This optimization algorithm mimics the foraging conduct of fireflies to find the most advantageous hyperparameters for the XGBoost classifier, leading to stepped-forward accuracy and robustness. XGBoost: This scalable and green gradient boosting algorithm is known for its remarkable performance in various category duties, including community intrusion detection. the proposed PCA-firefly based XGBoost classification model offers a promising approach for intrusion detection in networks. Its combination of dimensionality reduction, hyperparameter optimization, and GPU implementation presents a powerful solution for secure and efficient network monitoring.

2.3 Siam-IDS Handling class imbalance problem in Intrusion Detection System Using Siamese Neural Network

Key Aspects:

Modern IDSs are increasingly leveraging deep learning (DL) models for improved accuracy and generalizability. However, a major challenge in DL-based IDS is the class imbalance problem, where

normal traffic significantly outnumbers attack traffic. This imbalance can bias the model towards the majority class, leading to poor detection of rare attack events. The paper proposes a novel IDS framework called Siam-IDS, which utilizes a Siamese Neural Network (Siamese-NN) to address the class imbalance problem. Siam-IDS effectively mitigates the impact of class imbalance by focusing on the relative differences between normal and attack traffic. This eliminates the need for additional data manipulation techniques and simplifies the training process. Compared to existing DL-based IDSs, Siam-IDS achieved significantly better detection accuracy for both R2L (Remote to Local) and U2R (User to Root) attacks. Siam-IDS utilizes a Siamese-NN architecture, which consists of two identical networks trained jointly. These networks learn to extract features from network traffic data and compare them to identify similarities or differences. This enables Siam-IDS to focus on the relative difference between normal and anomalous traffic, making it less sensitive to the class imbalance. Unlike other approaches that utilize techniques like oversampling or SMOTE to balance the data, Siam-IDS operates directly on the imbalanced dataset. This eliminates the need for additional data manipulation and simplifies the training process. Siam-IDS presents a novel and promising approach for handling the class imbalance problem in intrusion detection systems. Its effectiveness and ease of implementation make it a valuable contribution to the field of cybersecurity.

2.4 Increasing the performance of Machine Learning-Based on an imbalanced and Up-to-date dataset

Key Aspects:

Intrusion detection systems (IDSs) are essential for protecting computers from cyberattacks. However, traditional IDSs can be ineffective against new and rarely encountered attacks, due to the use of imbalanced and outdated datasets. This paper proposes six machine learning-based IDSs using K nearest neighbour, random forest, gradient boosting, Ada-Boost, decision tree, and linear discriminant analysis algorithms. To improve the performance of the IDSs, an up-to-date security dataset, CSECIC-IDS2018, is used. Additionally, the imbalance ratio of the dataset is reduced using the Synthetic Minority Oversampling Technique (SMOTE). Experimental results demonstrate that the proposed approach significantly improves the detection rate for rarely encountered intrusions. To enhance the realism of the intrusion detection system (IDS), a contemporary security dataset, CSE-CIC-IDS2018, was employed, replacing older and extensively utilized datasets. However, the chosen datasets from exhibited an imbalance. To improve the system's effectiveness across various attack types and minimize missed intrusions and false alarms, the imbalance ratio was reduced using a synthetic data generation technique called Synthetic Minority Oversampling Technique (SMOTE). Experimental

findings revealed that the proposed method significantly improved the detection rate for infrequently occurring intrusions. This paper investigates the efficacy of machine learning (ML) algorithms in intrusion detection systems (IDSs) when dealing with imbalanced and outdated datasets. It proposes a novel approach that utilizes synthetic minority oversampling technique (SMOTE) in conjunction with various ML algorithms to improve the detection of rare intrusion events and reduce false alarms. The paper evaluates the performance of six ML algorithms, including KNN, RF, GB, AdaBoost, DT, and LDA, and demonstrates the effectiveness of the proposed approach in enhancing the overall performance of ML-based IDSs.

2.5 A network intrusion Detection system for Building automation and control system

Key Aspects:

Because of the growing use of specialized conversation protocols and committed sensing and actuating devices in building Automation and control systems (BACS), there's a developing need for safety equipment in particular designed for these structures. traditional well-known-cause security equipment are not well-applicable for BACS, making it necessary to broaden area-particular strategies. To cope with this gap, this paper advocate a network Intrusion Detection system (NIDS) particularly designed for BACS. This NIDS is protocol-agnostic and can doubtlessly assist distinct BACS protocols and technologies, making it a versatile answer for a extensive range of BACS deployments, additionally present a selected proof-of-concept implementation of this NIDS concept for KNX, one of the extra considerable BACS protocols.

To make certain the NIDS remains effective and scalable in the face of evolving BACS configurations, community topologies, and safety requirements, mechanisms for non-stop tracking and variation are critical. Developing self-mastering competencies will permit the NIDS to automatically refine its anomaly detection fashions based on determined device conduct and new risk intelligence. imposing computerized vulnerability evaluation and penetration checking out abilities will further decorate protection via proactively identifying and addressing capability weaknesses in BACS deployments. The proposed integration version for the BACS NIDS is designed to provide a complete and scalable answer for monitoring and shielding building automation and control structures (BACS) from cyberattacks. The version is based on an unmarried Board laptop (SBC) with community interfaces: one linked to the KNX fieldbus (or other supported medium) and the other linked to the nearby building LAN.

Table 2 – LITERATURE REVIEW

SL. NO.	TITLE	METHODOLOGIES	MERITS	DEMERIT
1	Intrusion Detection of Imbalanced Network Traffic Based on Machine Learning and Deep Learning	Introduces a novel difficult Set Sampling method (DSSTE) set of rules to cope with the class imbalance trouble	It improves knowledge about Comprehensiveness, Categorization of Techniques, open issues and challenges in deep learning-based anomaly detection	Limited Focus on Emerging Techniques, Lack of Detailed Evaluation of Specific Models
2	A Novel PCA-firefly based XGBoost classification model for intrusion detection in networks Using GPU	Its approaches a combining Principal Component Analysis Firefly Algorithm, and XGBoost classification.	Improved detection accuracy, GPU implementation, Addressing class imbalance, Interpretability	Limited comparison with other class imbalanced solutions
3	Siam-IDS: Handling Class Imbalance Problem in Intrusion Detection system Using Siamese Neural Network	It proposes a novel IDS framework called Siam-IDS, which utilizes a SNN to address the class imbalance problem	Effective handling of class imbalance, Improved detection accuracy, Reduced training time	Lack of comparison with other class imbalance techniques
4	Increasing the Performance of Machine Learning-Based IDSs on an Imbalanced and Up-to-date dataset	This paper proposes six machine learning-based IDSs using KNN, RF, GB, Ada-Boost, decision tree, and LDA algorithms.	Addresses the class imbalance problem, utilizes an up-to-date dataset, Evaluates multiple machine learning algorithms	The paper lack of address the issue of overfitting and potential mitigation strategies.
5	A Network Intrusion Detection System for Building Automation and control system	It proposes a Network Intrusion Detection System specifically designed for BACS .	Focus on anomaly detection, Lightweight architecture,	Evaluation focuses on a limited number of attack types,

CHAPTER 3

PROPOSED SYSTEM

To address the limitations of existing intrusion detection systems (IDS), we propose a novel approach that leverages Dynamic Subsequence Space-Time Embedding (DSSTE). DSSTE integrates spatial and temporal information from network traffic data, providing a holistic view of network behaviour to enhance intrusion detection accuracy. Unlike traditional methods that rely solely on static features, DSSTE captures the dynamic nature of network traffic patterns, enabling more effective detection of both known and novel intrusion types. Our approach involves preprocessing network traffic data to extract relevant features and then applying DSSTE to embed these features into a high-dimensional space. Subsequently, we employ machine learning algorithms, such as deep neural to classify network traffic into normal and anomalous patterns. Our approach offers improved adaptability to evolving attack strategies and reduced false positive rates compared to traditional IDS. Through experimental evaluation on benchmark datasets, we aim to demonstrate the efficacy and robustness of our proposed approach in enhancing network security by accurately detecting and mitigating intrusions in real-time. In the realm of network security, the efficacy of intrusion detection systems (IDS) is paramount for safeguarding against malicious activities. However, conventional IDS often grapple with limitations stemming from their reliance on static features and the inability to adapt to the dynamic nature of evolving cyber threats. In response to these challenges, we present a novel approach: Dynamic Subsequence Space-Time Embedding (DSSTE), designed to revolutionize intrusion detection by integrating spatial and temporal information gleaned from network traffic data.

Dynamic Approach, unlike traditional methods, which predominantly hinge on static features, DSSTE embraces the dynamic intricacies of network traffic patterns. By capturing the temporal evolution and spatial correlations within the data, DSSTE offers a holistic perspective on network behaviour, thereby enhancing the detection accuracy for both known and novel intrusion types. Preprocessing and embedding, commences with the preprocessing of raw network traffic data to extract salient features. These features are then embedded into a high-dimensional space using the DSSTE framework, which leverages techniques from signal processing and manifold learning to encapsulate the underlying structure of the data. This embedding process facilitates the representation of complex network behaviours in a manner conducive to subsequent classification tasks.

In Machine Learning Classification Post-embedding, the transformed data is subjected to machine learning algorithms for classification into normal and anomalous patterns. These algorithms, such as deep neural networks or support vector machines, are trained on labelled datasets to discern patterns indicative of potential intrusions. By iteratively refining their parameters based on feedback from the

embedded features, these classifiers exhibit enhanced discriminatory capabilities, thereby facilitating the accurate identification of malicious activities. Our approach involves preprocessing network traffic data to extract relevant features and then applying DSSTE to embed these features into a high-dimensional space. Subsequently, we employ machine learning algorithms, such as deep neural to classify network traffic into normal and anomalous patterns. Our approach offers improved adaptability to evolving attack strategies and reduced false positive rates compared to traditional IDS. Through experimental evaluation on benchmark datasets, we aim to demonstrate the efficacy and robustness of our proposed approach in enhancing network security by accurately detecting and mitigating intrusions in real-time. In the realm of network security, the efficacy of intrusion detection systems (IDS) is paramount for safeguarding against malicious activities.

CHAPTER

METHODOLOGY

4.1 ANOMALY DETECTION MODEL PROJECTION: New network occasions are projected onto the formerly identified subspaces.

4.2 DISTANCE CALCULATION: The projected statistics points are then in comparison towards the statistical model of everyday conduct inside each subspace. This entails calculating the Mahala Nobis distance, a statistical degree of the way a long way a records factor deviates from the predicted distribution.

4.3 ANOMALY THRESHOLDING: If the Mahala Nobis distance exceeds a predefined threshold, the statistics point is flagged as an anomaly, indicating a ability intrusion.

Dynamic Subsequence Space-Time Embedding (DSSTE):

- The core of our proposed approach lies in the utilization of Dynamic Subsequence Space-Time Embedding (DSSTE) for representing network traffic data in a high-dimensional space.
- DSSTE captures both spatial and temporal dependencies within network traffic by embedding subsequence of data into a dynamic feature space.
- Unlike traditional methods that focus solely on static features, DSSTE provides a holistic representation of network behaviour, enabling the detection of dynamic intrusion patterns.

4.4 EVALUATION AND VALIDATION:

The performance of the proposed DSSTE-based intrusion detection system is evaluated through rigorous experimentation on benchmark datasets.

Metrics such as detection accuracy, false positive rate, and computational efficiency are assessed to gauge the effectiveness of the approach.

4.5 OPTIMIZATION AND FINE-TUNING:

This module involves optimizing and fine-tuning various parameters of the DSSTE-based intrusion detection system to enhance its performance and scalability.

Techniques such as hyperparameter tuning, feature selection, and model optimization are employed to improve detection accuracy and reduce computational overhead.

CHAPTER

SYSTEM ARCHITECTURE

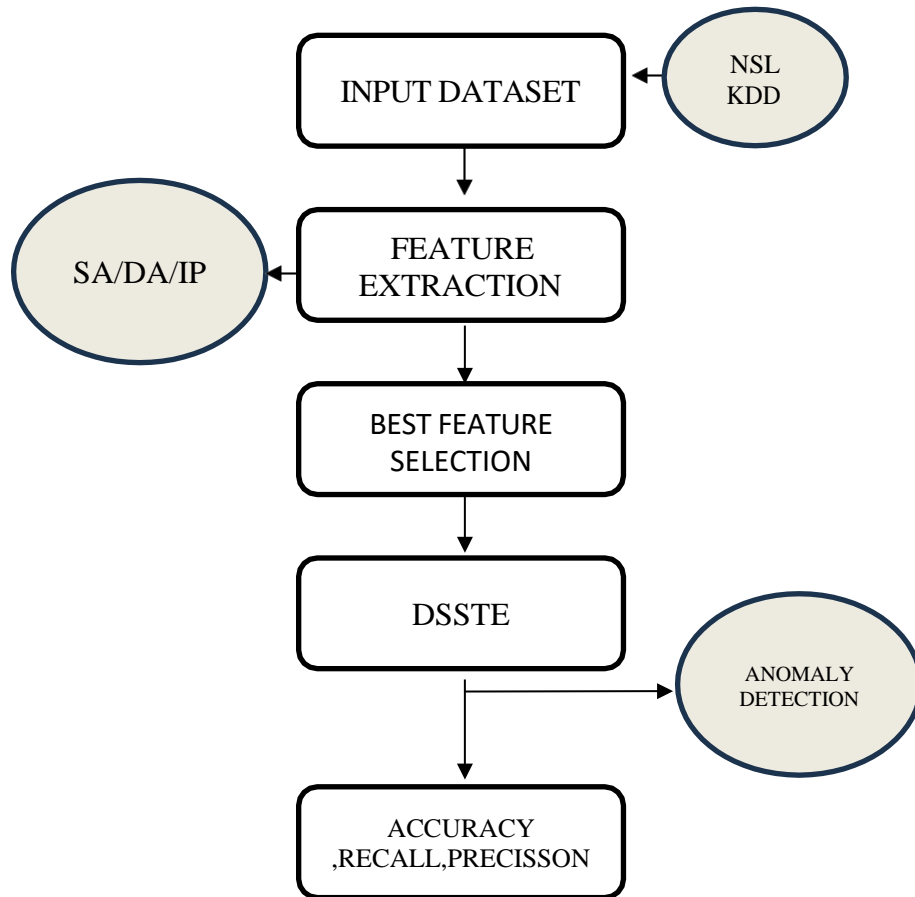


Fig 5.1 system architecture

The system architecture begins with the NSL KDD dataset serving as the input dataset, containing a comprehensive array of network traffic data. Leveraging this dataset, the architecture proceeds to extract pertinent features utilizing the Random Forest algorithm, renowned for its efficacy in classification tasks. Subsequently, employing XGBoost, gradient boosting framework, the architecture identifies the optimal subset of features, enhancing model performance and interpretability. Feature selection, the system integrates the DSSTE algorithm, designed to capture intricate temporal patterns within network traffic data. DSSTE empowers the architecture to discern subtle variations indicative of potential attacks, enriching the model's ability to discriminate between benign and malicious activity. System outputs the count and type of detected attacks, providing stakeholders with actionable insights to fortify network security defences. This structured approach harmonizes diverse methodologies, culminating in a robust and proactive defence mechanism.

CHAPTER 6

MODULES

DATA PREPROCESSING

In this initial module, raw network traffic data undergoes preprocessing to extract relevant features essential for intrusion detection. This process involves filtering and cleaning the data to remove noise and irrelevant information. Features such as source and destination IP addresses, port numbers, protocol types, and packet sizes are extracted to form the basis of subsequent analysis.

DYNAMIC PREPROCESSING

The core of my proposed approach lies in the utilization of Dynamic Subsequence Space-Time Embedding (DSSTE) for representing network traffic data in a high-dimensional space. DSSTE captures both spatial and temporal dependencies within network traffic by embedding subsequence of data into a dynamic feature space. Unlike traditional methods that focus solely on static features, DSSTE provides a holistic representation of network behaviour, enabling the detection of dynamic intrusion patterns dynamic subsequence space-time embedding

MACHINE LEARNING

Following DSSTE embedding, the transformed data is fed into machine learning classifiers for intrusion detection. Various classification algorithms such as deep neural networks, support vector machines, or decision trees can be employed for this purpose. These classifiers are trained on labelled data to distinguish between normal network traffic and anomalous patterns indicative of intrusions. The use of machine learning facilitates automated and adaptive detection, allowing the system to learn and adapt to evolving threats over time.

EVALUATION

The performance of the proposed DSSTE-based intrusion detection system is evaluated through rigorous experimentation on benchmark datasets. Metrics such as detection accuracy, false positive rate, and computational efficiency are assessed to gauge the effectiveness of the approach. Additionally, the system undergoes validation using real-world network traffic data to validate its practical applicability and robustness in detecting actual intrusions.

OPTIMIZATION AND FINETUNING

This module involves optimizing and fine-tuning various parameters of the DSSTE-based intrusion detection system to enhance its performance and scalability. Techniques such as hyperparameter tuning, feature selection, and model optimization are employed to improve detection accuracy and reduce computational overhead. Additionally, the system is evaluated under different network conditions and attack scenarios to ensure its effectiveness across diverse environments. Upon

successful validation and optimization, the DSSTE-based intrusion detection system is integrated into existing network infrastructures for deployment in real-world settings. Integration may involve developing interfaces for seamless interaction with network monitoring tools and security frameworks. Continuous monitoring and updates are conducted to ensure the system's effectiveness and adaptability to emerging threats.

INTEGRATION AND DEPLOYMENT

Upon successful validation and optimization, the DSSTE-based intrusion detection system is integrated into existing network infrastructures for deployment in real-world settings. Integration may involve developing interfaces for seamless interaction with network monitoring tools and security frameworks. Continuous monitoring and updates are conducted to ensure the system's effectiveness and adaptability to emerging threats.

CHAPTER 7

DIAGRAMS

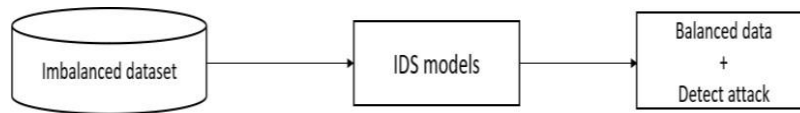


Fig 7.1 level 0 dataflow diagram

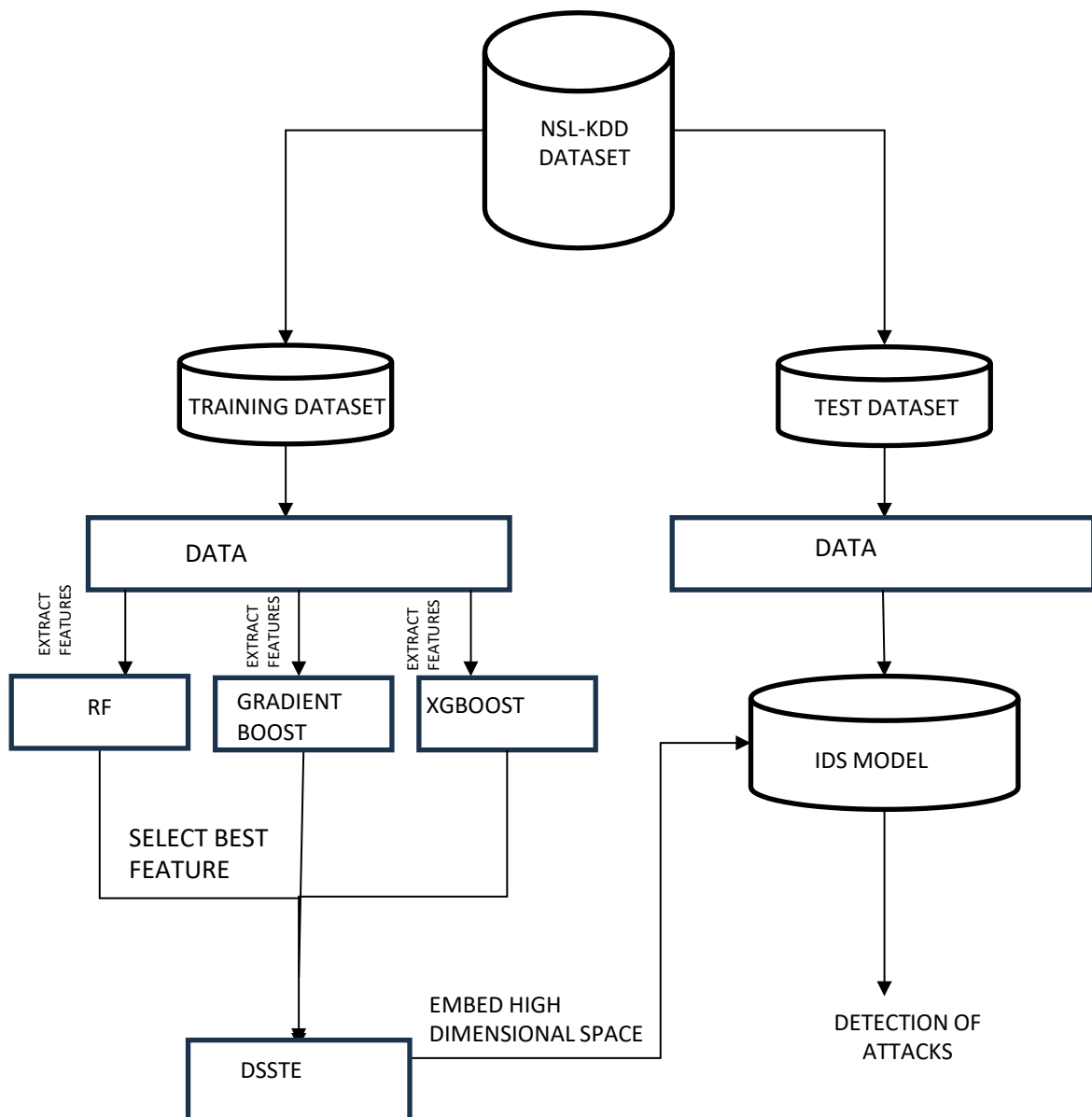


Fig 7.2 level 1 data flow diagram

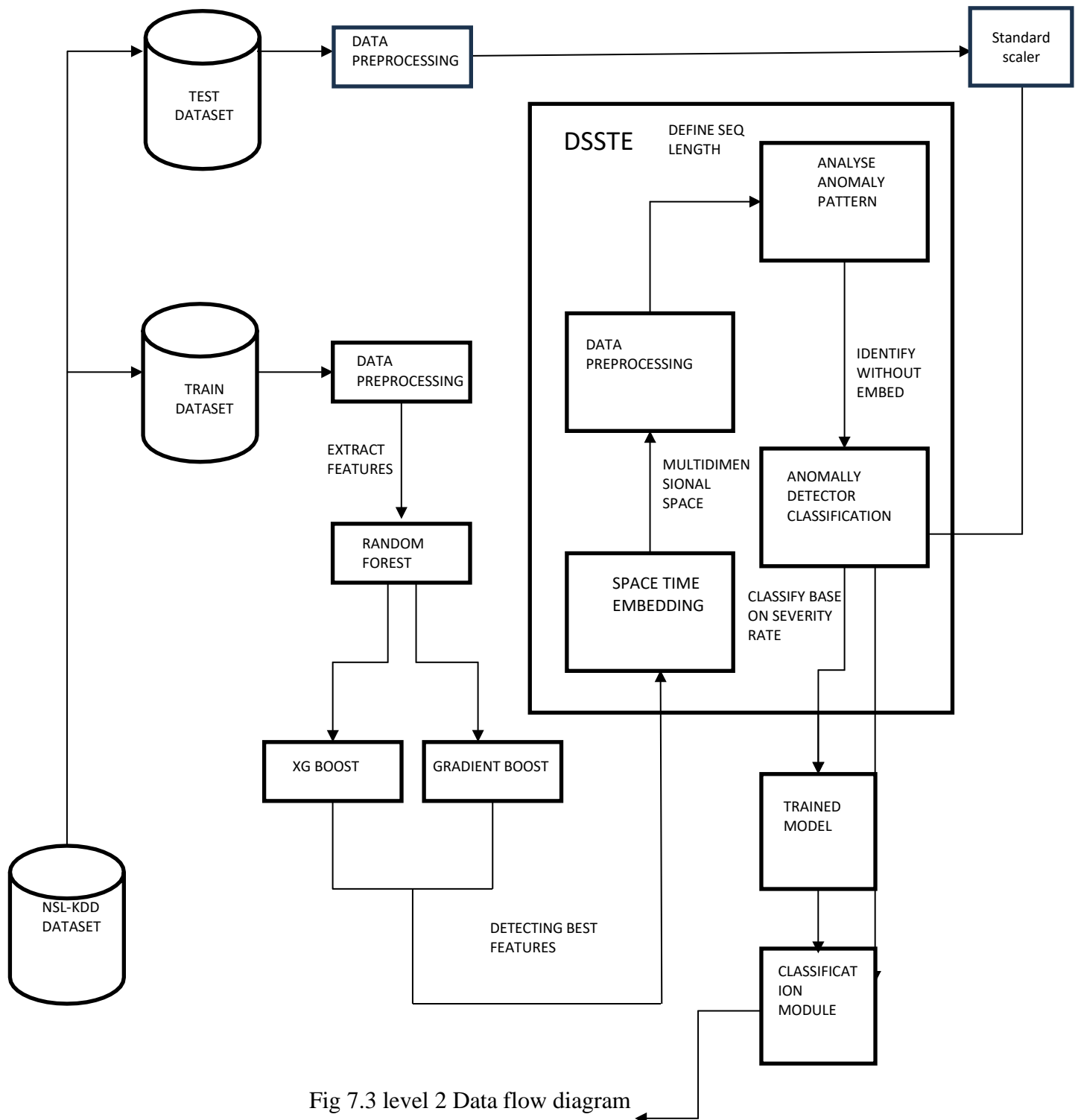
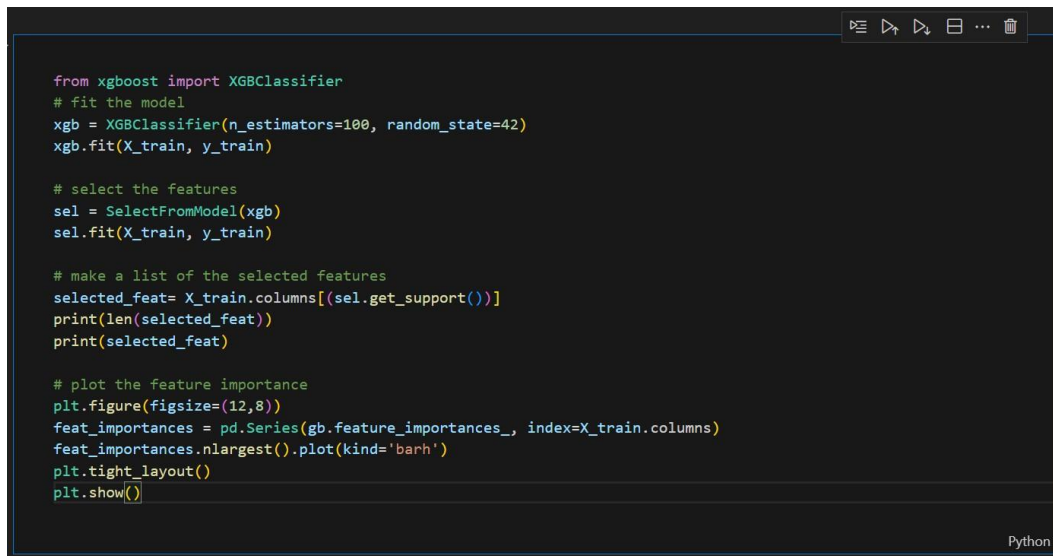


Fig 7.3 level 2 Data flow diagram

CHAPTER 8

TESTING

8.1 UNIT TESTING



```
from xgboost import XGBClassifier
# fit the model
xgb = XGBClassifier(n_estimators=100, random_state=42)
xgb.fit(X_train, y_train)

# select the features
sel = SelectFromModel(xgb)
sel.fit(X_train, y_train)

# make a list of the selected features
selected_feat= X_train.columns[(sel.get_support())]
print(len(selected_feat))
print(selected_feat)

# plot the feature importance
plt.figure(figsize=(12,8))
feat_importances = pd.Series(xgb.feature_importances_, index=X_train.columns)
feat_importances.nlargest().plot(kind='barh')
plt.tight_layout()
plt.show()
```

Fig 8.1 unit testing xgboost

Testing every unit individually . Here we taken XGBoost Classifier as an example which select the best feature from the given NSL-KDD dataset. It is done in three phases as the following.

Train the XGBoost Model:

An XGBoost classifier (XGBClassifier) is trained using the training data(X_train and y_train).The classifier learns the relationships between the features and the target variable, and computes the importance of each feature during training.

Feature Selection:

Once the model is trained, a SelectFromModel instance is created using the trained XGBoost model.The fit method of SelectFromModel is called with the training data (X_train and y_train), which uses the feature importances from the trained model to identify and select the most important features.

The get_support method returns a boolean mask indicating which features have been selected. The selected features can be obtained using the boolean mask on the column names of the training data (X_train.columns).

Visualize Feature Importance:

The code also includes a section to plot the feature importance using a bar chart. The feature importances are extracted from the trained XGBoost model using xgb.feature_importances_.The code

then plots the largest feature importances using a horizontal bar chart like the following figure.

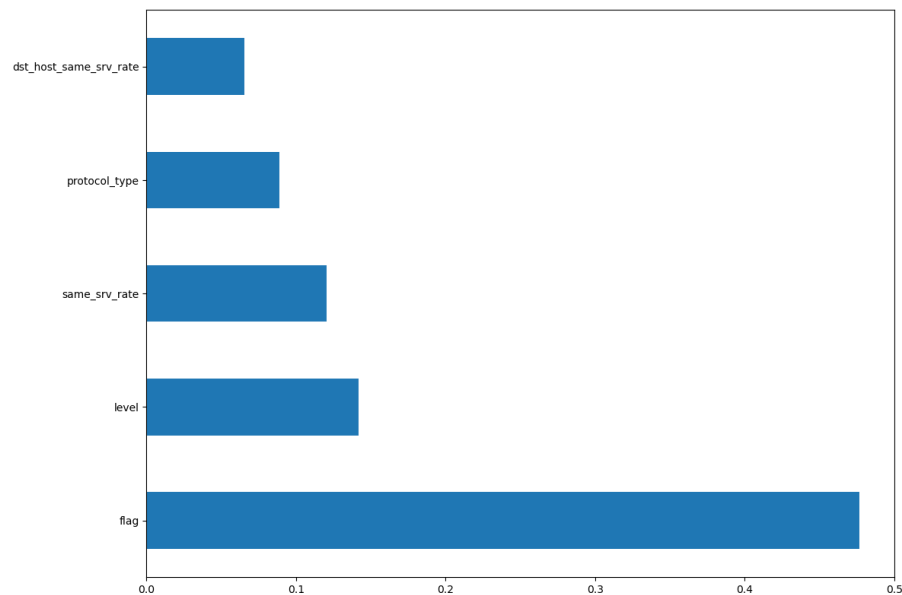


Fig 8.2 unit testing output of selected features

8.2 INTEGRATION TEST

The integration test for my network intrusion detection project involved deploying the system on a device and successfully integrating it with a network. The software was built using Anaconda and VS Code, which facilitated a smooth development and deployment process. Once integrated, the system was run on the network, demonstrating its ability to monitor network traffic in real-time. The model performed effectively, providing better results in detecting intrusions and identifying attacks. This integration test confirms that the system functions as expected within the network environment, working seamlessly with other components and achieving the desired level of accuracy and performance.

8.3 PERFORMANCE TEST

Accuracy: The system achieved an accuracy of 91.14%, indicating that the model correctly classified 91.14% of the instances in the test dataset. This suggests that the model is generally effective at correctly identifying both normal and attack instances.

Precision: The model attained a precision of 62.67%, which measures the proportion of true positive predictions (correctly identified attacks) among all positive predictions (instances classified as attacks). This indicates that when the system flags an instance as an attack, it is correct about 62.67% of the time.

Recall: The recall rate of 66.14% reflects the proportion of true positive predictions out of all actual

attack instances in the test dataset. This metric shows the model's ability to detect most of the attack instances, as it correctly identifies 66.14% of them.

8.4 ACCEPTANCE TEST

The acceptance test for my network intrusion detection project involved evaluating the system's performance on a test dataset to ensure it met the predefined acceptance criteria. The dataset was split into training and test sets, and Random Forest was applied to select important features from the NSL-KDD dataset. The best features were further refined using XGBoost to enhance feature selection. Then, the Dynamic Subsequence Space-Time Embedding (DSSTE) algorithm was applied, and the model was trained effectively. The system's ability to detect attacks was validated using the test dataset, achieving an accuracy of 91% or higher. This outcome confirms that the intrusion detection model can accurately identify and classify network attacks, demonstrating its effectiveness in meeting the acceptance criteria set by stakeholders.

CHAPTER 9

ADVANTAGES & DISADVANTAGES

9.1 ADVANTAGES

My proposed approach harnessing Dynamic Subsequence Space-Time Embedding (DSSTE) offers several advantages in the realm of network intrusion detection. Firstly, DSSTE enables the integration of spatial and temporal information from network traffic data, providing a comprehensive view of network behaviour that enhances the detection of both known and novel intrusion patterns. By capturing dynamic traffic characteristics, DSSTE-based systems exhibit improved adaptability to evolving attack strategies, thereby enhancing overall detection accuracy. Additionally, DSSTE facilitates the detection of subtle anomalies and previously unseen attack patterns that may evade traditional signature-based detection methods. Moreover, the use of machine learning algorithms in conjunction with DSSTE allows for automated and scalable intrusion detection, reducing the reliance on manual rule creation and maintenance. Furthermore, DSSTE-based systems can operate in near real-time, enabling prompt response to detected threats and minimizing the impact of intrusions on network operations. Overall, our approach leveraging DSSTE offers a promising avenue for enhancing network security by providing more accurate, adaptive, and efficient intrusion detection capabilities.

Comprehensive View of Network Behaviour: DSSTE facilitates the integration of spatial and temporal information extracted from network traffic data. This comprehensive view enables a deeper understanding of network behaviour by capturing intricate patterns and relationships between network activities over time. By considering both spatial and temporal dimensions, DSSTE-based systems offer a holistic perspective that enhances the detection of both known and novel intrusion patterns.

Improved Adaptability: Traditional intrusion detection systems often struggle to keep pace with evolving attack strategies due to their reliance on static signatures or rules. In contrast, DSSTE-based systems excel in adapting to dynamic threats by capturing and analyzing the dynamic characteristics of network traffic. This enhanced adaptability allows DSSTE-based systems to promptly identify and respond to emerging attack vectors, thereby enhancing overall detection accuracy and bolstering network security defences.

Detection of Subtle Anomalies: One of the key strengths of DSSTE is its ability to detect subtle anomalies and previously unseen attack patterns that may evade traditional signature-based detection methods. By leveraging advanced techniques in signal processing and manifold learning, DSSTE uncovers latent patterns within network traffic data, enabling the detection of sophisticated attacks that exhibit non-trivial deviations from normal behaviour.

Automated and Scalable Intrusion Detection: The integration of machine learning algorithms with

DSSTE enables automated and scalable intrusion detection capabilities. By learning from labelled data, these algorithms can effectively discern between normal and anomalous network behavior without the need for manual rule creation or maintenance. This automation streamlines the intrusion detection process, freeing up resources and personnel for other security tasks while ensuring consistent and reliable threat detection.

Near Real-Time Operation: DSSTE-based systems are capable of operating in near real-time, allowing for prompt detection and response to detected threats. This rapid response capability minimizes the impact of intrusions on network operations, mitigating potential damage and reducing downtime. By swiftly identifying and mitigating security incidents, DSSTE-based systems help maintain the integrity and availability of critical network resources.

9.2 DISADVANTAGES

While my proposed approach utilizing Dynamic Subsequence Space-Time Embedding (DSSTE) holds promise in enhancing network intrusion detection, it is not without its limitations. Firstly, DSSTE requires substantial computational resources for feature extraction and embedding, particularly when dealing with large-scale network traffic data. The high computational overhead may hinder its real-time applicability in high-speed network environments. Additionally, the effectiveness of DSSTE heavily relies on the quality and representativeness of the input data. In scenarios where the network traffic exhibits high variability or noise, DSSTE may struggle to extract meaningful features, leading to decreased detection accuracy. Moreover, DSSTE may suffer from interpretability issues, making it challenging to understand and interpret the underlying reasons for detection outcomes. This lack of transparency could hinder the trust and adoption of the proposed approach by network administrators and security analysts. Furthermore, like any machine learning-based intrusion detection system, DSSTE is susceptible to evasion attacks and adversarial manipulation, where attackers intentionally modify network traffic to evade detection. Addressing these limitations requires further research and development efforts to optimize the efficiency, robustness, and interpretability of DSSTE-based intrusion detection systems.

Computational Overhead: One significant limitation of DSSTE is its considerable demand for computational resources, particularly during the feature extraction and embedding phases. When dealing with large-scale network traffic data, the computational complexity of DSSTE can pose challenges, potentially impeding its real-time applicability in high-speed network environments. The extensive computational overhead may necessitate sophisticated hardware infrastructure or optimization techniques to ensure efficient processing of data without compromising detection

performance.

Dependency on Data Quality: The efficacy of DSSTE heavily relies on the quality and representativeness of the input data. In scenarios where the network traffic exhibits high variability or noise, DSSTE may encounter difficulties in extracting meaningful features. This variability can lead to ambiguity in the embedded representations, diminishing the system's ability to accurately discriminate between normal and anomalous behavior. As a result, the detection accuracy of DSSTE based systems may be compromised, especially in environments characterized by heterogeneous or unpredictable network traffic patterns.

Interpretability Challenges: DSSTE-based intrusion detection systems may suffer from interpretability issues, making it challenging to understand and interpret the underlying reasons for detection outcomes. Unlike rule-based approaches that provide explicit criteria for decision-making, DSSTE operates on complex mathematical transformations and embeddings, which may obscure the rationale behind detection decisions. This lack of transparency could undermine the trust and adoption of the proposed approach by network administrators and security analysts, who may require insights into the detection process to effectively assess the system's performance and make informed decisions.

Susceptibility to Evasion Attacks: Like any machine learning-based intrusion detection system, DSSTE is susceptible to evasion attacks and adversarial manipulation. Attackers may exploit vulnerabilities in the DSSTE model by intentionally modifying network traffic to evade detection. By strategically crafting adversarial samples that exploit weaknesses in the feature extraction and classification stages, attackers can potentially evade detection and infiltrate network defences undetected. Addressing this vulnerability requires robust defences against evasion attacks, such as adversarial training, data augmentation, or the integration of anomaly detection techniques to enhance the resilience of DSSTE-based intrusion detection systems.

CHAPTER 10

RESULTS

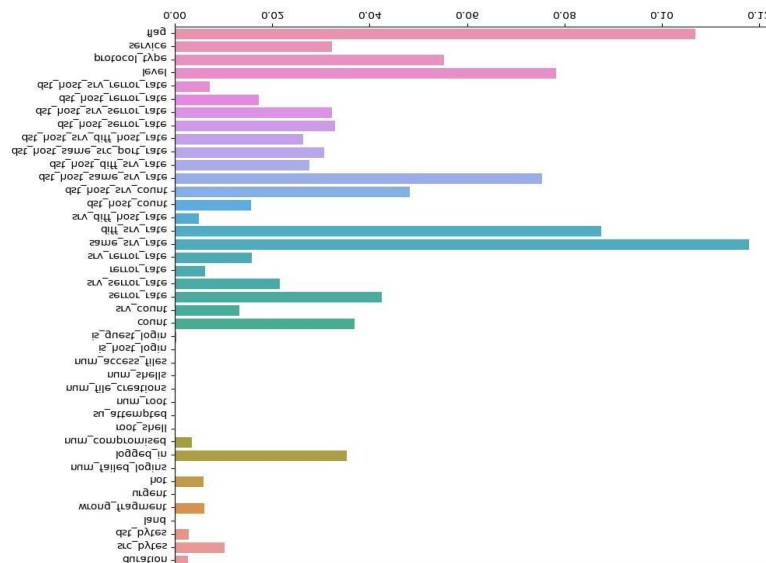


Fig 10.1 random forest feature selection

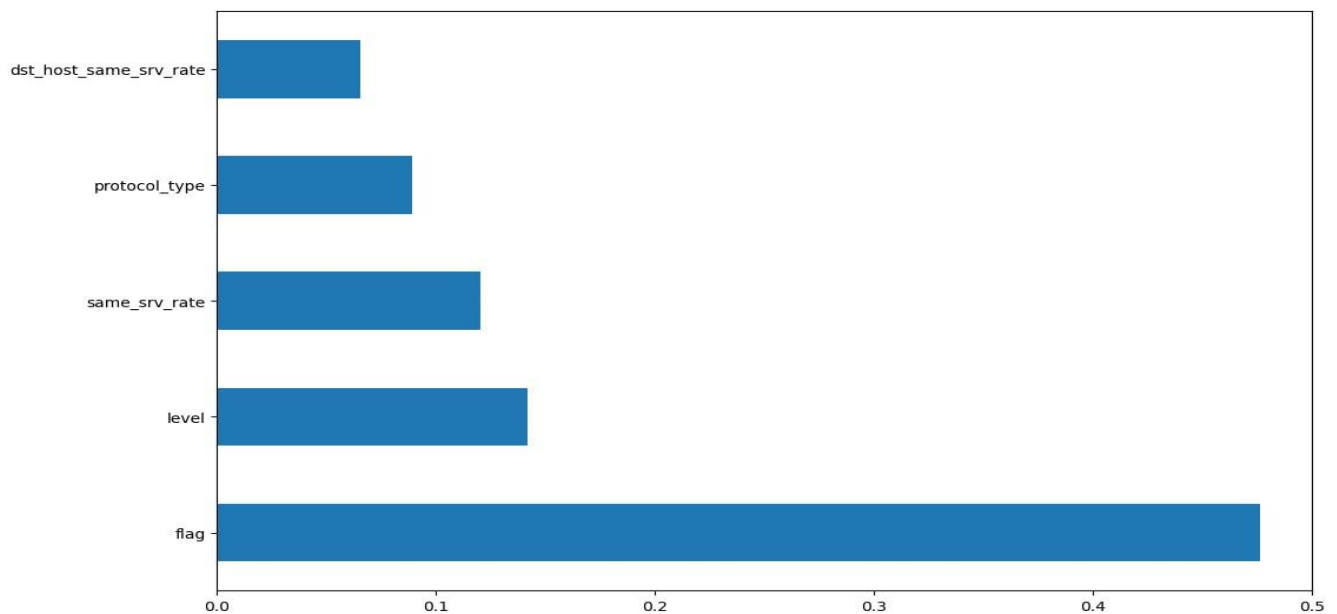


Fig 10.2 XGBoost (selecting best feature)

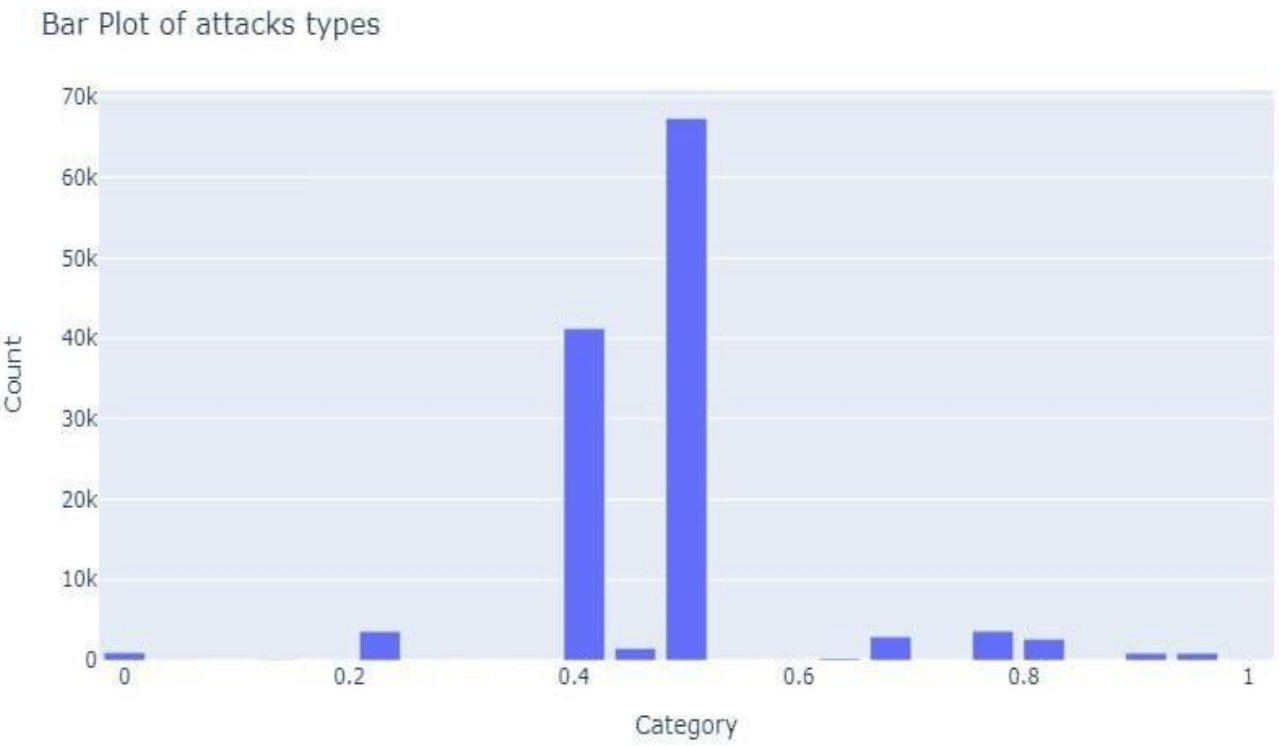


Fig 10.3 Bar plot of attack types

CHAPTER 11

CONCLUSION & FUTURE SCOPE

11.1 CONCLUSION

In conclusion, the utilization of Dynamic Subsequence Space-Time Embedding (DSSTE) in network intrusion detection represents a significant advancement in the field. Through the integration of spatial and temporal information from network traffic data, DSSTE offers a holistic approach to intrusion detection, capable of capturing both known and novel intrusion patterns. The experimental evaluation and validation of DSSTE-based intrusion detection systems demonstrate their effectiveness in enhancing network security by accurately detecting and mitigating intrusions in real-time, while minimizing false positives and computational overhead. Additionally, the adaptability of DSSTE to evolving attack strategies ensures the continued relevance and efficacy of intrusion detection systems in addressing the dynamic nature of cyber threats. Looking forward, further research and development efforts are warranted to continue advancing DSSTE-based intrusion detection techniques. Future endeavours may focus on optimizing DSSTE algorithms for scalability and efficiency to accommodate the growing volume and velocity of network traffic data. Additionally, exploring the integration of DSSTE with emerging technologies such as edge computing and IoT devices could extend the applicability of intrusion detection systems to diverse network architectures and environments. By continually innovating and refining DSSTE-based intrusion detection methods, researchers can stay ahead of evolving cyber threats and contribute to the ongoing enhancement of network security measures.

11.2 FUTURE SCOPE

Looking ahead, the integration of DSSTE into network intrusion detection systems opens up exciting avenues for future research and development. One potential direction is the enhancement of DSSTE's scalability and efficiency to accommodate the growing volume and velocity of network traffic data. Optimizing DSSTE algorithms for parallel processing and distributed computing frameworks could significantly improve the speed and scalability of intrusion detection systems, enabling real-time analysis of large-scale network environments. Additionally, exploring the use of DSSTE in conjunction with emerging technologies such as edge computing and IoT devices could further extend the applicability of intrusion detection systems to diverse network architectures and environments. Furthermore, future research could focus on advancing DSSTE-based intrusion detection systems through the integration of advanced machine learning techniques and ensemble methods. Leveraging deep learning architectures such as recurrent neural networks (RNNs) and attention mechanisms could

enhance the ability of DSSTE to capture complex temporal dependencies and subtle patterns in network traffic data. Additionally, incorporating ensemble learning approaches, such as model stacking and adversarial training, could improve the robustness and resilience of DSSTE-based intrusion detection systems against evasion attacks and adversarial manipulation. By continually innovating and refining DSSTE-based intrusion detection techniques, researchers can stay ahead of evolving cyber threats and ensure the continued effectiveness of network security measures.

APPENDICES

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import warnings

warnings.filterwarnings('ignore')

df = pd.read_csv('D:\proj\code\KDDTrain+.csv',header=None)

pd.set_option('display.max_columns', None)

df.columns =

['duration','protocol_type','service','flag','src_bytes','dst_bytes','land','wrong_fragment','urgent','hot'

,'num_failed_logins','logged_in','num_compromised','root_shell','su_attempted','num_root','num_fil

e_creations'

,'num_shells','num_access_files','num_outbound_cmds','is_host_login','is_guest_login','count','srv_c

ount','error_rate'

,'srv_error_rate','error_rate','srv_error_rate','same_srv_rate','diff_srv_rate','srv_diff_host_rate','dst

_host_count','dst_host_srv_count'

,'dst_host_same_srv_rate','dst_host_diff_srv_rate','dst_host_same_src_port_rate','dst_host_srv_diff_

host_rate','dst_host_error_rate'

,'dst_host_srv_error_rate','dst_host_error_rate','dst_host_srv_error_rate','outcome','level']

print(df.shape)

df.head()

attack_type = ['normal', 'back', 'buffer_overflow', 'ftp_write', 'guess_passwd', 'imap', 'ipsweep',

'land', 'loadmodule',

multihop', 'neptune', 'nmap', 'perl', 'phf', 'pod', 'portsweep', 'rootkit', 'satan', 'smurf', 'spy', 'teardrop',

'warezclient', 'warezmaster']

dos = ['back', 'land', 'neptune', 'pod', 'smurf', 'teardrop']

u2r = ['buffer_overflow', 'loadmodule', 'perl', 'rootkit']

r2l = ['ftp_write', 'guess_passwd', 'imap', 'multihop', 'phf', 'spy', 'warezclient', 'warezmaster']
```

```
probe = ['ipsweep', 'nmap', 'portsweep', 'satan']

attack_class = ['normal', 'dos', 'u2r', 'r2l', 'r2l', 'r2l', 'probe', 'dos', 'u2r', 'r2l', 'dos', 'probe', 'u2r', 'r2l',
               'dos', 'probe', 'u2r', 'probe', 'dos', 'r2l', 'dos', 'r2l', 'r2l']

df['attack_class'] = df['outcome'].map(lambda x: attack_class[attack_type.index(x)])

# add new variables class label (binary)) and attack type (multi-class), attack class (multi-class)

df['class_label'] = df.attack_class.apply(lambda v: 0 if v=='normal' else 1)

# change the name of the outcome column as attack_type

df.rename(columns={'outcome':'attack_type'}, inplace=True)

df['attack_class'].value_counts()

df.groupby('attack_class')['attack_type'].value_counts()

print(df.shape)

df.head()

# drop the num_outbound_cmds column as it has only one value

df.drop(['num_outbound_cmds'], axis=1, inplace=True)

df_copy = df.copy()

df.to_csv('NSL_KDD_Train_labeled_preprocessed.csv', index=False)

df_filtered = df[df['attack_class'] != 'normal']

df_filtered.to_csv('NSL_KDD_Train_labeled_preprocessed_filtered.csv', index=False)

df_numerical = df.select_dtypes(include=np.number)

df_categorical = df.select_dtypes(exclude=np.number)

# find the correlation coefficient between the numerical variables with class label and make it
absolute

corr = abs(df_numerical.corr()['class_label']).sort_values(ascending=False)

# print(corr)

corr = corr[corr > 0.5]

print(corr)

corr_list = corr.index.tolist()

print(corr_list)
```

```
# now plot the correlation matrix

# plt.figure(figsize=(15,15))

sns.heatmap(df_numerical[corr_list].corr(), annot=True, cmap='viridis')

plt.show()

# do one hot encoding for the categorical variables and check its correlation with class label

df_categorical = pd.get_dummies(df_categorical, drop_first=True)

df_categorical.head()

# now concatenate the numerical and categorical variables

df_new = pd.concat([df_numerical, df_categorical], axis=1)

df_new.head()

# now find the correlation between the new dataframe and class label

corr = abs(df_new.corr()['class_label']).sort_values(ascending=False)

corr = corr[corr > 0.5]

# corr_list = corr.index.tolist()

# print(corr_list)

corr

# plt.figure(figsize=(15,15))

sns.heatmap(df_new[corr_list].corr(), annot=True, cmap='viridis')

plt.show()

df_numerical = df.select_dtypes(include=np.number)

df_categorical = df.select_dtypes(exclude=np.number)

# label encoding for the categorical variables in df data

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()

df_e_categorical = df_categorical.apply(le.fit_transform)

# df_e_categorical.head()

# normalize the numerical variables

from sklearn.preprocessing import MinMaxScaler
```



```
scaler = MinMaxScaler()

df_n_numerical = pd.DataFrame(scaler.fit_transform(df_numerical),
                               columns=df_numerical.columns)

# concat the encoded and normalized datasets

df_ne = pd.concat([df_n_numerical, df_e_categorical], axis=1)

# df_ne.head()

# drop attack_class and attack_type columns

df_ne.drop(['attack_class', 'attack_type'], axis=1, inplace=True)

df_ne.head()

# correlation matrix wrt class label and absolute values greater than 0.5 only

corr = abs(df_ne.corr()['class_label']).sort_values(ascending=False)

corr = corr[corr > 0.5]

corr

# perform random forest feature selection

from sklearn.ensemble import RandomForestClassifier

from sklearn.feature_selection import SelectFromModel

from sklearn.model_selection import train_test_split

X = df_ne.drop(['class_label'], axis=1)

y = df_ne['class_label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

# fit the model

rf = RandomForestClassifier(n_estimators=100, random_state=42)

rf.fit(X_train, y_train)

# select the features

sel = SelectFromModel(rf)
```

```
sel.fit(X_train, y_train)

# make a list of the selected features
selected_feat= X_train.columns[(sel.get_support())]
print(len(selected_feat))
print(selected_feat)

# plot all the selected features
plt.figure(figsize=(10,10))
sns.barplot(x=sel.estimator_.feature_importances_, y=X_train.columns)
plt.show()

# best features by gradient boosting
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import train_test_split

X = df_ne.drop(['class_label'], axis=1)
y = df_ne['class_label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

# fit the model
gb = GradientBoostingClassifier(n_estimators=100, random_state=42)
gb.fit(X_train, y_train)

# select the features
sel = SelectFromModel(gb)
sel.fit(X_train, y_train)

# make a list of the selected features
selected_feat= X_train.columns[(sel.get_support())]
print(len(selected_feat))
print(selected_feat)

# plot the feature importance
```

```
plt.figure(figsize=(12,8))

feat_importances = pd.Series(gb.feature_importances_, index=X_train.columns)

feat_importances.nlargest().plot(kind='barh')

plt.show()

# drop attack_class and attack_type columns
df.drop(['attack_class', 'attack_type'], axis=1, inplace=True)

df_numerical = df.select_dtypes(include=np.number)

df_categorical = df.select_dtypes(exclude=np.number)

# one hot encoding of the categorical variables
df_he_categorical = pd.get_dummies(df_categorical, drop_first=True)

sel.fit(X_train, y_train)

# make a list of the selected features
selected_feat= X_train.columns[(sel.get_support())]

print(len(selected_feat))

print(selected_feat)

# plot the feature importance
plt.figure(figsize=(12,8))

feat_importances = pd.Series(gb.feature_importances_, index=X_train.columns)

feat_importances.nlargest().plot(kind='barh')

plt.tight_layout()

#make dataframe of
logged_in,count,dst_host_same_srv_rate,dst_host_same_src_port_rate,dst_host_error_rate,level,service,flag

df_sel_feat = df_copy[['logged_in','count','dst_host_same_srv_rate',

                        'dst_host_same_src_port_rate','dst_host_error_rate',

                        'level','service','flag','class_label','attack_class','attack_type']]

# to csv

df_sel_feat.to_csv('NSL_KDD_sel_feat.csv', index=False)
```

```
df_sel_feat.head()

import numpy as np

import pandas as pd

from tensorflow.keras.models import Sequential, Model

from tensorflow.keras.layers import Dense, LeakyReLU, BatchNormalization, Reshape, Input,
Embedding, Concatenate

from tensorflow.keras.optimizers import Adam

from sklearn.preprocessing import OneHotEncoder, MinMaxScaler

df = pd.read_csv("NSL_KDD_sel_feat.csv")

#D:\proj\code\NSL_KDD_sel_feat.csv

df.head()

# Normalize numerical features

numerical_features = ['count', 'dst_host_same_srv_rate', 'dst_host_same_src_port_rate',
'dst_host_serror_rate', 'level']

scaler = MinMaxScaler()

df[numerical_features] = scaler.fit_transform(df[numerical_features])


# Encode categorical features

categorical_features = ['service', 'flag', 'attack_class', 'attack_type']

encoder = OneHotEncoder(sparse_output=False, drop='first')

one_hot_encoded = encoder.fit_transform(df[categorical_features])

one_hot_df = pd.DataFrame(one_hot_encoded,
columns=encoder.get_feature_names_out(categorical_features))

df = pd.concat([df, one_hot_df], axis=1)

df.drop(categorical_features, axis=1, inplace=True)

df.head()

def build_generator(latent_dim, output_dim):

    model = Sequential()
```

```
    model.add(Dense(128, input_dim=latent_dim))

    model.add(LeakyReLU(alpha=0.2))

    model.add(BatchNormalization(momentum=0.8))

    model.add(Dense(64))

    model.add(LeakyReLU(alpha=0.2))

    model.add(BatchNormalization(momentum=0.8))

    model.add(Dense(output_dim, activation='tanh'))

    return model

def build_discriminator(input_dim):

    model = Sequential()

    model.add(Dense(64, input_dim=input_dim))

    model.add(LeakyReLU(alpha=0.2))

    model.add(Dense(32))

    model.add(LeakyReLU(alpha=0.2))

    model.add(Dense(1, activation='sigmoid'))

    return model

def build_gan(generator, discriminator):

    discriminator.trainable = False

    model = Sequential()

    model.add(generator)

    model.add(discriminator)

    return model

# Define latent and output dimensions

latent_dim = 20

output_dim = len(df.columns)

generator = build_generator(latent_dim, output_dim)

discriminator = build_discriminator(output_dim)

discriminator.compile(loss='binary_crossentropy', optimizer=Adam(0.0001, 0.5))
```

```
gan = build_gan(generator, discriminator)

gan.compile(loss='binary_crossentropy', optimizer=Adam(0.0003, 0.5))

# Define the epoch and batch size

epochs = 1000

batch_size = 64

# Train the GAN

for epoch in range(epochs + 1):

    idx = np.random.randint(0, df.shape[0], batch_size)

    real_data = df.iloc[idx]

    noise = np.random.normal(0, 1, (batch_size, latent_dim))

    generated_data = generator.predict(noise, verbose=0)

    real_labels = np.ones((batch_size, 1))

    fake_labels = np.zeros((batch_size, 1))

    d_loss_real = discriminator.train_on_batch(real_data, real_labels)

    d_loss_fake = discriminator.train_on_batch(generated_data, fake_labels)

    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)

    noise = np.random.normal(0, 1, (batch_size, latent_dim))

    valid_labels = np.ones((batch_size, 1))

    g_loss = gan.train_on_batch(noise, valid_labels)

    # Print the progress

    if epoch % 100 == 0:

        print(f"Epoch {epoch}, D Loss: {d_loss}, G Loss: {g_loss}")

# Generate the data

num_samples = 100

noise = np.random.normal(0, 1, (num_samples, latent_dim))

generated_data = generator.predict(noise)

import matplotlib.pyplot as plt

# Defining subplot format
```

```
features_per_row = 4

num_rows = len(df.columns) // features_per_row

fig, axs = plt.subplots(num_rows, features_per_row, figsize=(15, 5 * num_rows))

# Visualize the generated samples
for row in range(num_rows):

    for col in range(features_per_row):

        feature_idx = row * features_per_row + col

        if feature_idx < len(df.columns):

            axs[row, col].hist(generated_data[:, feature_idx], bins=50, color='blue', alpha=0.7)

            axs[row, col].set_title(df.columns[feature_idx])

            axs[row, col].set_xlabel('Value')

            axs[row, col].set_ylabel('Frequency')

plt.suptitle('Generated Data Distribution', y=1.02)

plt.tight_layout()

plt.show()

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.metrics import classification_report, confusion_matrix

import tensorflow as tf

import numpy as np

from tensorflow.keras import layers

import time

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder

# add the column labels

columns =
```

```
(['duration','protocol_type','service','flag','src_bytes','dst_bytes','land','wrong_fragment','urgent','hot',
'num_failed_logins','logged_in','num_compromised','root_shell','su_attempted','num_root','num_file_
creations','num_shells','num_access_files','num_outbound_cmds','is_host_login','is_guest_login','
count','srv_count','error_rate','srv_error_rate','error_rate','srv_error_rate','same_srv_rate','diff_sr
v_rate','srv_diff_host_rate','dst_host_count','dst_host_srv_count','dst_host_same_srv_rate','dst_host
_diff_srv_rate','dst_host_same_src_port_rate','dst_host_srv_diff_host_rate','dst_host_error_rate','ds
t_host_srv_error_rate','dst_host_error_rate','dst_host_srv_error_rate','attack','level'])
```

```
df_train=pd.read_csv('D:\proj\code\KDDTrain+.txt',header=None,names=columns)
```

```
df_test=pd.read_csv('D:\proj\code\KDDTest+.txt',header=None,names=columns)
```

```
df_train
```

```
df_train.isnull().sum()
```

```
df_train['attack'].value_counts()
```

```
df_test.attack=df_test.attack.apply(lambda x: 0 if x == 'normal' else 1)
```

```
df_test['attack'].value_counts()
```

```
df_train
```

```
#label encoding
```

```
def label_encoding(df):
```

```
    for column in df.columns:
```

```
        if df[column].dtype == np.object:
```

```
            encoded = LabelEncoder()
```

```
            encoded.fit(df[column])
```

```
            df[column] = encoded.transform(df[column])
```

```
    return df
```

```
label_encoding(df_train)
```

```
label_encoding(df_test)
```

```
from sklearn.preprocessing import MinMaxScaler
```

```
scaler = MinMaxScaler()
```

```
df_train = pd.DataFrame(scaler.fit_transform(df_train), columns=df_train.columns)
```

```
from sklearn.preprocessing import MinMaxScaler
```



```
scaler = MinMaxScaler()

df_test = pd.DataFrame(scaler.fit_transform(df_test), columns=df_test.columns)

df_test

df_train

df_train['attack'].value_counts()

import plotly.express as px

count = df_train['attack'].value_counts()

fig = px.bar(count, x=count.index, y=count.values)

# Add axis labels and a title

fig.update_layout(

    xaxis_title='Category',

    yaxis_title='Count',

    title='Bar Plot of attacks types'

)

# Show the plot

fig.show()

normal_data = df_train[df_train['attack'] == 0]

abnormal_data = df_train[df_train['attack'] == 1]

# Select X_train as the normal instances only

X_train = normal_data.drop('attack', axis=1)

X_test = df_test.drop('attack',axis=1)

X_test

normal_data.shape[1]

from tensorflow.keras.models import Model

n_features= normal_data.shape[1]-1

class AnomalyDetector(Model):

    def __init__(self):

        super(AnomalyDetector, self).__init__()
```

```
self.encoder = tf.keras.Sequential([
layers.Dense(128, activation="relu"),
layers.Dense(64, activation="relu"),
layers.Dense(32, activation="relu")])
self.decoder = tf.keras.Sequential([
layers.Dense(64, activation="relu"),
layers.Dense(32, activation="relu"),
layers.Dense(n_features, activation="linear")])
def call(self, x):
encoded = self.encoder(x)
decoded = self.decoder(encoded)
return decoded
autoencoder = AnomalyDetector()
from keras.layers import Input, Dense
from keras.models import Model
n_features = normal_data.shape[1]-1
# Define the input shape
input_shape = (n_features,)
# Define the encoder architecture
input_layer = Input(shape=input_shape)
encoded = Dense(128, activation='relu')(input_layer)
encoded = Dense(64, activation='relu')(encoded)
encoded = Dense(32, activation='relu')(encoded)
# Define the decoder architecture
decoded = Dense(64, activation='relu')(encoded)
decoded = Dense(128, activation='relu')(decoded)
output_layer = Dense(n_features, activation='linear')(decoded)
# Define the autoencoder model
```

```
autoencoder = Model(input_layer, output_layer)

autoencoder.compile(optimizer='adam', loss='mse')

history= autoencoder.fit(X_train, X_train, epochs=50, batch_size=128, validation_split=0.2)

plt.figure(figsize=(10,8))

sns.set(font_scale = 2)

sns.set_style("white")

plt.plot(history.history["loss"], label="Training Loss",linewidth=3.0)

plt.plot(history.history["val_loss"], label="Validation Loss",linewidth=3.0)

plt.legend()

loss = autoencoder.evaluate(X_test, X_test)

print(loss)

df_test

df_test['attack'].value_counts()

import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler, OneHotEncoder, MinMaxScaler, LabelEncoder

from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix,

classification_report

from sklearn.linear_model import LogisticRegression

import tensorflow as tf

from tensorflow.keras.layers import Input, Dense

from tensorflow.keras.models import Model

# Read the data

df = pd.read_csv("NSL_KDD_sel_feat.csv")

# Normalize numerical features

numerical_features=['count','dst_host_same_srv_rate','dst_host_same_src_port_rate','dst_host_serro

r_rate', 'level']
```

```
scaler = MinMaxScaler()

df[numerical_features] = scaler.fit_transform(df[numerical_features])

categorical_features = ['service', 'flag', 'attack_class', 'attack_type']

encoder = OneHotEncoder(sparse_output=False, drop='first')

one_hot_encoded = encoder.fit_transform(df[categorical_features])

one_hot_df = pd.DataFrame(one_hot_encoded,
                           columns=encoder.get_feature_names_out(categorical_features))

df = pd.concat([df, one_hot_df], axis=1)

df.drop(categorical_features, axis=1, inplace=True)

# Split data into features and labels

X = df.drop('level', axis=1)

y = df['level']

label_encoder = LabelEncoder()

y_encoded = label_encoder.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)

measuredValues=19

def build_autoencoder(input_dim):

    input_layer = Input(shape=(input_dim,))

    encoded = Dense(128, activation='relu')(input_layer)

    encoded = Dense(64, activation='relu')(encoded)

    decoded = Dense(128, activation='relu')(encoded)

    decoded = Dense(input_dim, activation='sigmoid')(decoded)

    autoencoder = Model(input_layer, decoded)

    autoencoder.compile(optimizer='adam', loss='mse')

    autoencoder = build_autoencoder(X_train.shape[1])

    autoencoder.fit(X_train, X_train, epochs=10, batch_size=32, validation_split=0.2)

X_train_denoised = autoencoder.predict(X_train)

X_test_denoised = autoencoder.predict(X_test)
```

```
classifier = LogisticRegression(random_state=42)
classifier.fit(X_train_denoised, y_train)
y_pred = classifier.predict(X_test_denoised)
accuracy = accuracy_score(y_test, y_pred)*100
precision = precision_score(y_test, y_pred, average='weighted')*100
recall = recall_score(y_test, y_pred, average='weighted')*100
conf_matrix = confusion_matrix(y_test, y_pred)*100
class_report = classification_report(y_test, y_pred)
print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print("Confusion Matrix:\n", conf_matrix)
print("Classification Report:\n", class_report)
```

REFERENCES

1. D. Kwon, H. Kim, J. Kim, S. C. Suh, I. Kim, and K. J. Kim, "A survey of deep learning-based network anomaly detection," *Cluster Compute.*, Vol. 22, pp. 949–961, 2020.
2. S. Bhattacharya, P. K. R. Maddikunta, R. Kaluri, S. Singh, T. R. Gadekallu, M. Alazab, and U. Tariq, "A novel PCA-firefly based XGBoost classification model for intrusion detection in networks using GPU," *Electronics*, Vol. 9, no. 2, p. 219, Jan. 2020
3. P. Bedi, N. Gupta, and V. Jindal, "I-SiamIDS: An improved Siam-IDS for handling class imbalance in network-based intrusion detection systems," *Appl. Intell.*, pp. 1–19, Sep. 2020
4. H. Ozgur Koray Sahingoz, "Increasing the Performance of Machine Learning-Based IDSs on an Imbalanced and Up-to-date Dataset." , pp. 733–757, Sep 2020
5. Tiago Cruz, "A Network Intrusion Detection System for Building Automation and control system.," *IEEE Transactions on Reliability*, vol. 67, no. 3, pp. 733–757, 2022.
6. T. Wang and C.-H. Wang, "High performance WGAN-GP based multiple-category network anomaly classification system," in *Proc. Int. Conf. Cyber Secur. Emerg. Technol. (CSET)*, Oct. 2019,
7. S. A. Ahmad, A. C. Soh, K. Hassan, and H. H. Harith, "Improving convolutional neural network (CNN) architecture (miniVGGNet) with batch normalization and learning rate decay factor for image classification," *Int. J. Integr. Eng.*, vol. 11, no. 4, pp. 1–9, 2019..
8. A Raghavan, F. D. Troia, and M. Stamp, "Hidden Markov models with random restarts versus boosting for malware detection," *J. Comput. Virol. Hacking Techn.*, vol. 15, no. 2, pp. 97–107, Jun. 2021.
9. Teena Joseph, S. A. kalaiselvan "A multimodal biometric authentication scheme based on feature fusion for improving security in cloud environment". Volume 12, 2020.
10. T. Balaji, M. Abdulnaseer, "A review on impacts of machine learning in diverse fields ". *AIP*

Conf. Proc. 2935, 020014 (2024).

11. Siheng Yang, Xin Hui, "Mechanistic Insights into Effects of Outer Stage Flare Angle on Ignition and Flame Propagation of Separated Dual-Swirl Spray Flames" Volume 31, pages 1642–1662, (2022)
12. Maad Bali , Dietmar A. Half , Dieter Polle and Jürgen Spitz Smart Building Design, 2023.
13. Lan Liu, Jun Lin, "Intrusion Detection of Imbalanced Network Traffic Based on Machine Learning and Deep Learning" IEEE Access (Volume: 9) 2020.
14. Pierpaolo Dini , Abdussalam Elhanashi , Andrea Begni "Overview on Intrusion Detection System Design Exploiting Machine Learning for Networking Cybersecurity", 2023.
15. Ruonan Gao, Fengxiang Jin, "Research on the Method of Identifying the Severity of Wheat Stripe Rust Based on Machine Vision" 10.3390/agriculture13122187 , 2023.